```
    {                                   // 查找范围非空
        mid = (l + r) / 2; // 取中间位置的元素作为基准值
        if (a[mid] >= x)
        { // 基准值符合要求,区间右端点向左调整
            res = mid;
            r = mid - 1;
        }
        else
            l = mid + 1; // 基准值不符合要求,区间左端点向右调整
    }
    return res;
}
```

参考词条

算法模板库中的常用函数

延伸阅读

王晓东．计算机算法设计与分析[M]．3 版．北京：电子工业大学出版社，2007：15-16.

典型题目

1. NOIP2001 提高组 一元三次方程求解
2. NOIP2011 提高组 聪明的质检员
3. NOIP2012 提高组 借教室
4. NOIP2015 提高组 跳石头
5. USACO17JAN Cow Dance Show

（陈奕哲　金靖　谢秋锋）

1.4.3.5 倍增法

倍增法(binary lifting method)是一种通过成倍增长的方式来优化问题求解过程的算法，其基本思想是先通过成倍增长的方式求出状态空间上 2 的整数次幂项位置的值，再利用这些值组合为需要求解的答案。

以 x^n 的计算为例，若 $n=2^{b_1}+2^{b_2}+\cdots+2^{b_w}(b_1<b_2<\cdots<b_w)$，则根据幂运算的运算法可得 $x^n=x^{2^{b_1}+2^{b_2}+\cdots+2^{b_w}}=x^{2^{b_1}}\times x^{2^{b_2}}\times\cdots\times x^{2^{b_w}}$。此时令 $y=x^{2^0}$ 并重复执行 b_w 次 $y=y\times y$ 操作，便可计算出 $x^{2^1},x^{2^2},x^{2^3},\cdots,x^{2^{b_w}}$ 的所有值，随后最多执行 b_w 次乘法操作可得答案。由于 n 的二进制表示最多有 $\lfloor \log_2 n \rfloor+1$ 个非零位，故运用倍增法计算 x^n 的时间复杂度为 $O(\log n)$。倍增法优化的幂运算也被称为快速幂。

代码示例

倍增法计算 x^n 的核心代码如下。

```
long long quick_pow(long long x,int n)
{
    long long res = 1;
    while (n > 0)
    { // 若 n 大于 0,说明还有等于 1 的二进制位需要处理
        if (n & 1)
            res = res * x; // n&1 算出二进制下最后一位,若该位的值为 1 就将当前位对
                           应的 x 的 2^k 次幂的值乘入答案
    // 由于后续不断执行 n>>=1 操作,此处 n&1 的结果依次为初始 n 的二进制位上第 1,2,…位
        x = x * x; // 倍增计算 x 的 2^k 次幂,每操作一次 k 的值就 +1
        n >>= 1; // 处理完当前位后,通过右移操作消去当前位
    }
    return res;
}
```

参考词条

1. ST 表
2. 最近公共祖先

典型题目

1. NOIP2013 提高组 转圈游戏
2. NOIP2012 提高组 开车旅行

（陈奕哲　谢秋锋）

1.4.4 数值处理算法

1.4.4.1 高精度加法

数值大小超出计算机标准数据类型可表示范围的数被称为高精度数。高精度加法是指实现高精度数加法操作的算法，算法的基本思路是将高精度数按数位拆分后分别存储，再通过模拟加法竖式来计算结果。高精度加法的主要步骤如下：

（1）以字符串形式读入两个高精度数，分别表示两个加数；

（2）将两个高精度数按数位拆分后，逆序存储在两个数组中；

（3）模拟加法竖式，按位分别计算相加的结果；

（4）从低位到高位依次处理进位的情况；

（5）从高位到低位依次输出。

代码示例

高精度加法的核心代码如下。

```
scanf("%s%s",s1 + 1,s2 + 1); // 读入两个高精度数
len1 = strlen(s1 + 1),len2 = strlen(s2 + 1);
len3 = max(len1,len2); // 计算高精度数的长度
for (int i = 1; i <= len1; i++)
    num1[i] = s1[len1 - i + 1] - '0'; // 按数位拆分后逆序存储
for (int i = 1; i <= len2; i++)
    num2[i] = s2[len2 - i + 1] - '0';
for (int i = 1; i <= len3; i++)
{
    res[i] += num1[i] + num2[i];
    if (res[i] >= 10)
    { // 处理进位
        res[i + 1]++;
        res[i] -= 10;
    }
}
if (res[len3 + 1])
    len3++;
for (int i = len3; i >= 1; i--)
    printf("%d",res[i]); // 从高位到低位依次输出

struct bignum
{
    char s[20005];
    // len 表示长度,a 数组按位存储高精度数拆分后的值,flag 表示数字的正负状态
    int len,a[20005],flag;
    // 无参构造,初始化为 0,长度为 1,非负
    bignum()
    {
        memset(a,0,sizeof(a));
        len = 1;
        flag = 0;
    }
    // 有参构造,将单精度数 x 以高精度数的方式存储
    bignum(int x)
    {
        for (len = 1; x; len++)
        {
            a[len] = x %10;
            x /= 10;
```

```
        }
        len--;
    }
    // 重载 [],使调用每一位的值时更方便
    int &operator[](int i)
    {
        return a[i];
    }
    // 重载 <,用于比较两个高精度数的大小
    friend bool operator<(bignum a,bignum b)
    {
        if (a.len < b.len)
            return true;
        if (a.len > b.len)
            return false;
        for (int i = a.len; i >= 1; i--)
        {
            if (a[i] < b[i])
                return true;
            if (a[i] > b[i])
                return false;
        }
        return false;
    }
    // 定义高精度数的输入函数
    void input()
    {
        scanf("%s",s + 1);
        len = strlen(s + 1);
        for (int i = 1; i <= len; i++)
            a[i] = s[len - i + 1] - '0';
    }
    // 定义高精度数的输出函数
    void print()
    {
        if (flag)
            putchar('-');
        for (int i = len; i >= 1; i--)
            printf("%d",a[i]);
    }
};
// 重载高精度加法
bignum operator+(bignum a,bignum b)
{
```

```
    bignum res;
    res.len = max(a.len,b.len);
    for (int i = 1; i <= res.len; i++)
    {
        res[i] += a[i] + b[i];
        if (res[i] >= 10)
        {
            res[i + 1]++;
            res[i] -= 10;
        }
    }
    if (res[res.len + 1])
        res.len++;
    return res;
}
// 高精度数的加法计算
bignum a,b,c;
a.input();
b.input();
c = a + b;
c.print();
```

（陈奕哲　谢秋锋）

1.4.4.2　高精度减法

高精度减法是指实现高精度数减法操作的算法，算法的基本思路是将高精度数按数位拆分后分别存储，再通过模拟减法竖式来计算结果。高精度减法的主要步骤如下：

（1）以字符串形式读入两个高精度数，分别表示被减数和减数，若减数大于被减数，则最后输出时先输出一个负号，同时交换被减数和减数；

（2）将两个高精度数按数位拆分后，逆序存储在两个数组中；

（3）模拟减法竖式，按位分别计算相减的结果；

（4）从低位到高位处理借位的情况；

（5）从高位到低位依次输出（前导 0 不输出）。

代码示例

高精度减法的核心代码如下。

```
scanf("%s%s",s1 + 1,s2 + 1); // 读入两个高精度数
len1 = strlen(s1 + 1);
len2 = strlen(s2 + 1);
f = false;
if (len1 < len2 || (len1 == len2 && strcmp(s1 + 1,s2 + 1) < 0))
```